NAME

dir, **dirent** - directory file format

SYNOPSIS

#include <dirent.h>

DESCRIPTION

Directories provide a convenient hierarchical method of grouping files while obscuring the underlying details of the storage medium. A directory file is differentiated from a plain file by a flag in its inode(5) entry. It consists of records (directory entries) each of which contains information about a file and a pointer to the file itself. Directory entries may contain other directories as well as plain files; such nested directories are referred to as subdirectories. A hierarchy of directories and files is formed in this manner and is called a file system (or referred to as a file system tree).

Each directory file contains two special directory entries; one is a pointer to the directory itself called dot '.' and the other a pointer to its parent directory called dot-dot '..'. Dot and dot-dot are valid pathnames, however, the system root directory '/', has no parent and dot-dot points to itself like dot.

File system nodes are ordinary directory files on which has been grafted a file system object, such as a physical disk or a partitioned area of such a disk. (See mount(2) and mount(8).)

The directory entry format is defined in the file *<sys/dirent.h>* (which should not be included directly by applications):

```
#ifndef    _SYS_DIRENT_H_
#define    _SYS_DIRENT_H_

#include <machine/ansi.h>

/*
 * The dirent structure defines the format of directory entries returned by
 * the getdirentries(2) system call.
 *
 * A directory entry has a struct dirent at the front of it, containing its
 * inode number, the length of the entry, and the length of the name
 * contained in the entry.  These are followed by the name padded to a 8
 * byte boundary with null bytes.  All names are guaranteed null terminated.
 * The maximum length of a name in a directory is MAXNAMLEN.
 * Explicit pad is added between the last member of the header and
 * d_name, to avoid having the ABI padding in the end of dirent on
```

```
 * LP64 arches.  There is code depending on d_name being last.  Also,
 * keeping this pad for ILP32 architectures simplifies compat32 layer.
 */

struct dirent {
        ino_t      d_fileno;            /* file number of entry */
        off_t      d_off;               /* directory offset of the next entry */
        __uint16_t d_reclen;                    /* length of this record */
        __uint8_t  d_type;              /* file type, see below */
        __uint8_t  d_namlen;                    /* length of string in d_name */
        __uint32_t d_pad0;
#if __BSD_VISIBLE
#define    MAXNAMLEN     255
        char       d_name[MAXNAMLEN + 1];           /* name must be no longer than this */
#else
        char       d_name[255 + 1];   /* name must be no longer than this */
#endif
};

/*
 * File types
 */
#define    DT_UNKNOWN    0
#define    DT_FIFO               1
#define    DT_CHR        2
#define    DT_DIR        4
#define    DT_BLK        6
#define    DT_REG        8
#define    DT_LNK        10
#define    DT_SOCK               12
#define    DT_WHT                14

/*
 * Convert between stat structure types and directory types.
 */
#define    IFTODT(mode)      (((mode) & 0170000) >> 12)
#define    DTTOIF(dirtype)   ((dirtype) << 12)

/*
 * The _GENERIC_DIRSIZ macro gives the minimum record length which will hold
```

```
 * the directory entry.  This returns the amount of space in struct direct
 * without the d_name field, plus enough space for the name with a terminating
 * null byte (dp->d_namlen+1), rounded up to a 8 byte boundary.
 *
 * XXX although this macro is in the implementation namespace, it requires
 * a manifest constant that is not.
 */
#define    _GENERIC_DIRLEN(namlen)                                          ((__offsetof(struct di
#define    _GENERIC_DIRSIZ(dp)      _GENERIC_DIRLEN((dp)->d_namlen)
#endif /* __BSD_VISIBLE */

#ifdef _KERNEL
#define    GENERIC_DIRSIZ(dp)       _GENERIC_DIRSIZ(dp)
#endif

#endif /* !_SYS_DIRENT_H_ */
```

SEE ALSO

fs(5), inode(5)

HISTORY

A **dir** file format appeared in Version 7 AT&T UNIX.

BUGS

The usage of the member d_type of struct dirent is unportable as it is FreeBSD-specific. It also may fail on certain file systems, for example the cd9660 file system.